Linear Regression

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model. Before attempting to fit a linear model to observed data, a modeler should first determine whether or not there is a relationship between the variables of interest. This does not necessarily imply that one variable causes the other (for example, higher SAT scores do not cause higher college grades), but that there is some significant association between the two variables. A scatterplot can be a helpful tool in determining the strength of the relationship between two variables. If there appears to be no association between the proposed explanatory and dependent variables (i.e., the scatterplot does not indicate any increasing or decreasing trends), then fitting a linear regression model to the data probably will not provide a useful model. A valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1 indicating the strength of the association of the observed data for the two variables.

A linear regression line has an equation of the form Y = a + bX, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when x = 0).

Create a random variable sequence("x") of size 100

```
import random
np.random.seed(1)
#numpy.random.seed(seed=None)
#Seed the generator.
#This method is called when RandomState is initialized. It can be called again to re-se
x=np.random.randn(100)
#here we tell the generator that the size is 100
#RandomState exposes a number of methods for generating random numbers drawn from a var
print (x)
```

```
[ 1.62434536 -0.61175641 -0.52817175 -1.07296862  0.86540763 -2.3015387
  1.74481176 -0.7612069   0.3190391  -0.24937038  1.46210794 -2.06014071
 -0.3224172  -0.38405435  1.13376944 -1.09989127 -0.17242821 -0.87785842
  0.04221375  0.58281521 -1.10061918  1.14472371  0.90159072  0.50249434
  0.90085595 -0.68372786 -0.12289023 -0.93576943 -0.26788808  0.53035547
 -0.69166075 -0.39675353 -0.6871727  -0.84520564 -0.67124613 -0.0126646
 -1.11731035  0.2344157   1.65980218  0.74204416 -0.19183555 -0.88762896
 -0.74715829  1.6924546   0.05080775 -0.63699565  0.19091548  2.10025514
  0.12015895  0.61720311  0.30017032 -0.35224985 -1.1425182  -0.34934272
 -0.20889423  0.58662319  0.83898341  0.93110208  0.28558733  0.88514116
 -0.75439794  1.25286816  0.51292982 -0.29809284  0.48851815 -0.07557171
  1.13162939  1.51981682  2.18557541 -1.39649634 -1.44411381 -0.50446586
  0.16003707  0.87616892  0.31563495 -2.02220122 -0.30620401  0.82797464
  0.23009474  0.76201118 -0.22232814 -0.20075807  0.18656139  0.41005165
  0.19829972  0.11900865 -0.67066229  0.37756379  0.12182127  1.12948391
```

Create a random variable sequence("y") of size 100,

where y = random_sequence + 2*x + 1

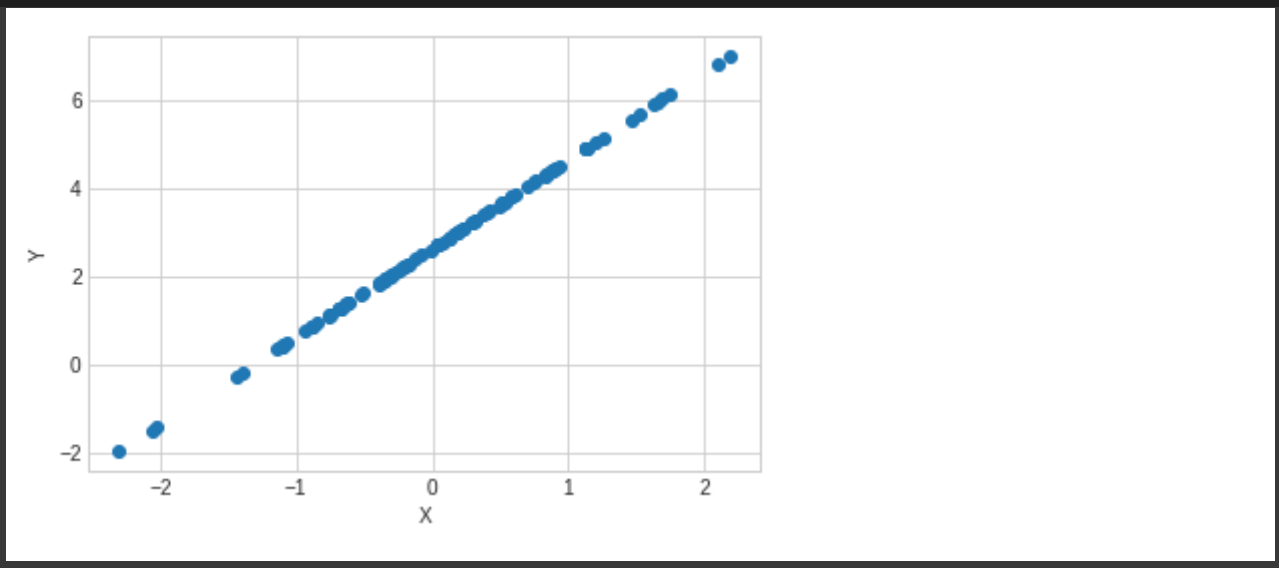
```
np.random.seed(1)
#here we initialise the seed generator like we did in the x variable above with 1
#and also form the y vaiable according to the condition given
y = np.random.randn(1) + 2*x +1
print (y)
```

```
[ 5.87303609  1.40083254  1.56800186  0.47840812  4.35516062 -1.97873203
  6.11396889  1.10193156  3.26242356  2.12560461  5.54856124 -1.49593606
  1.97951096  1.85623665  4.89188425  0.42456283  2.27948895  0.86862853
  2.70877286  3.78997579  0.42310701  4.91379278  4.4275268   3.62933404
  4.42605726  1.25688965  2.37856491  0.7528065   2.0885692   3.6850563
  1.24102386  1.83083831  1.24999996  0.93393408  1.2818531   2.59901617
  0.38972467  3.09317676  5.94394972  4.10843368  2.24067426  0.84908744
  1.13002878  6.00925457  2.72596087  1.35035407  3.00617633  6.82485564
  2.86466327  3.85875158  3.224686    1.91984567  0.33930897  1.92565992
  2.2065569   3.79759175  4.30231219  4.48654953  3.19552001  4.39462769
  1.11554948  5.13008167  3.650205    2.02815969  3.60138166  2.47320194
  4.88760414  5.663979    6.99549618 -0.16864731 -0.26388225  1.61541364
  2.9444195   4.37668321  3.25561526 -1.42005707  2.01193734  4.28029465
  3.08453483  4.14836772  2.17968908  2.22282923  2.99746815  3.44444866
  3.0209448   2.86236266  1.28302079  3.37947294  2.86798791  4.88331318
  5.02218112  2.9946582   1.87377546  1.34688455  3.47133407  2.7790255
  1.93663801  2.71153908  1.38434368  4.02040943]
```

Plot the scatter plot between x & y.

```
import matplotlib.pyplot as plt
#matplotlib.pyplot is a collection of command style functions that make matplotlib work
#Each pyplot function makes some change to a figure: eg, create a figure, create a plot
# plot some lines in a plotting area, decorate the plot with labels, etc....
#matplotlib.pyplot is stateful, in that it keeps track of the current figure and plotti
plt.scatter(x,y)
#Scatter plots are used to plot data points on horizontal and vertical axis in the atte
plt.xlabel("X")
plt.ylabel("Y")
```

```
10 plt.show()
```



Find pearson correlation between x & y

```
1 import scipy.stats
2 print("Pearson Correlation Coefficient:",scipy.stats.pearsonr(x,y)[0])
3 #it is one of the many corelational fucntions used in scipy.stats module
4 #pearsonr(x, y)
5 #Pearson correlation coefficient and p-value for testing non-correlation.
6 #This module contains a large number of probability distributions as well as a growing
7
8
```

```
Pearson Correlation Coefficient: 0.9999999999999999
```